PureResponse API User Guide

Version 2.8, published 22 March 2013 11:11

Revision History

Version	Date	Author	Description
0.1	04/02/2009	Greg Dreyfus	Initial version
1.6	19/01/2010	Greg Dreyfus	WSDL URL revision
1.7	08/03/2010	Greg Dreyfus	Opt-Out URL updated.
1.8	19/05/2010	Greg Dreyfus	List Sign-Up used as Opt-Out form updated.
1.9	02/06/2010	Greg Dreyfus	Added BLOCKED trigger type
2.0	29/07/2010	Greg Dreyfus	Updated Event Notifications & Direct Opt-Out
2.1	05/01/2011	Greg Dreyfus	Add FTP Batch Upload as Event Notification Destination
2.2	25/01/2011	Greg Dreyfus	Removed reference to “web source pages”
2.3	18/11/2011	Greg Dreyfus	Updated Event Notifications via FTP Batch Upload
2.4	15/02/2012	Greg Dreyfus	Added note to web request method
2.5	03/05/2012	Greg Dreyfus	Updated Event Notification information
2.6	14/11/2012	Chris Northwood	Updated Event Notification information
2.7	20/03/2013	Hugh Wooller	Updated Sign-up Interface information Updated Auto List Upload information Updated Event Notification information
2.8	21/03/2013	Chris Northwood	Updated Event Notification information

Contents

1 About This Document

We recognise that email may be only a small part of your overall marketing mix, and that it would be best if our software fitted into your other marketing processes as closely as possible.

We're therefore committed to providing a variety of methods to allow you to integrate your email marketing process into your own systems.

Examples of integration projects might include:

- Using a web form to add an email address and/or other details to a list held within PureResponse.
- Creating emails from content held on your own content management system.
- If you are reading this document then you are probably considering creating an interface between PureResponse and your software application or website. This document should help explain the options to you and give you an idea of the sort of technical skills you will require.
- If you are unsure how to achieve your requirement using the interfaces described here, then we can offer an analysis service to define your requirements and suggest the best solution for you. We can also offer a comprehensive software development service on request.

2 Integration Summary

The following provides a brief summary of the different integration options available within PureResponse.

2.1 From you to PureResponse

Here is a summary of the different ways of sending data into PureResponse other than using the website.

List sign-Up

Adding, editing and opting out one record in a list.

Using this interface you can set-up your website or your web application to add a record to one of your lists. You can also update an existing record's personalised data (see next) or unsubscribe the record entirely.

List sign-Up preload

Pre-populating a web page with a recipient's personalised data.

If you've created a sign-up page to allow a recipient to join your list, then you can use this interface to pre-populate that page with the recipient's data. They can then edit their data and update PureResponse. This is especially useful for adding an "edit my preferences" type link to your emails.

Automated list upload

Letting your web applications upload your lists.

This interface allows you to directly link your system to PureResponse and automatically create, replace, and append to, lists within your account.

Post-click tracking

Let PureResponse report on what happens after they leave the email.

This interface allows you to record what happens on your website after the recipient has clicked on a link in your email, and then see the results inside PureResponse.

PAINT

Pure360 Application INTerface.

PAINT allows your systems to perform many of the functions that are available within PureResponse e.g. creating messages, uploading lists, scheduling deliveries, and accessing reports. It uses SOAP to handle requests and responses.

2.2 From PureResponse to you

Here is a summary of the different ways that PureResponse can send data to you other than using the website.

Event notifications

Receive updates on bounces, opt-outs, opens and more.

These notifications can be sent via email, over the web to an application, or to automatically update CRM systems such as Salesforce and MS Dynamics.

They can be sent when a recipient bounces, opts in, opts out, opens or clicks a link. They can also be sent when your campaigns start and finish.

SMS reply module

Send a response when an SMS is received.

You can receive an email, SMS or web notification when an SMS is received to your keyword and shortcode.

3 List Sign-Up

Using this interface you can set-up your website or your web application to add a record to one of your lists. You can also update an existing record's personalised data or unsubscribe the record entirely.

3.1 Add and edit a single record

This interface adds a new recipient record to an existing list within PureResponse. The email address or mobile passed in is what makes the record unique.

If you pass both email and mobile then PureResponse will create two new records. If a record with the same email or mobile number already exists, then the existing record is updated.

Important

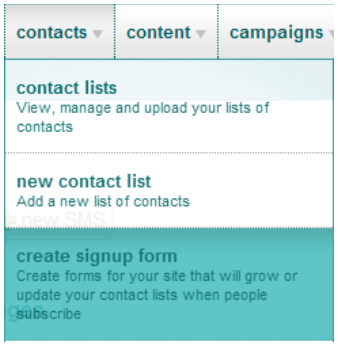
When you add a new email record to your profile using this interface, we automatically send it a double opt-in confirmation message. The record will only be included in deliveries from PureResponse after the recipient has followed the link in the email that confirms that they are opting in. Until they do so their status will remain limited.

This process is designed to protect you and the recipient from SPAM and we strongly recommend that you use it. However, if you decide that you do not want to use this facility, please see the "advanced options" section later in the chapter.

You can use this interface from a simple web page or directly from your application. You'll find the following sections useful once you've decided which approach is best for you.

- **Via a web page**

If you are using it from a web page then we recommend that you log into the PureResponse system and access the *create signup form* section. There you will find a form you can fill in to automatically generate an example sign-up page.



You'll be asked to complete a few pieces of information when you set-up your sign-up page.

Here are some descriptions to help you:-

Field Name	Description
success url	Set this to the full address (http://...) of the page on your web site that should be displayed after a record has been successfully added to the list. This should normally be a “thank you" type page. If you have any problems setting up your sign-up page then you may find it useful to set this field to `NO-REDIRECT`. You’ll then be able to see any error messages being generated. Don’t forget to set it back again once everything is working.
error url	Set this to the full address (http://...) of the page on your web site that should be displayed if a record fails to be added because of either validation or system errors. You’ll probably want this to be a “sorry, there’s something wrong" type page, possibly with a back button. If you want to give your user some specific error messages, then if you place the text `{~error~}` on this page, PureResponse will replace it with the specific error message before showing it to the user.

- **Direct from your application**

You may have a comprehensive web application already, and want to capture the sign-up data in your database first. One way to do this is to modify your application so that it receives the data first and then sends a web request directly to PureResponse passing in all the data required.

This type of request is much the same as the HTML form and we would recommend that you follow the process described above first. Once you have a working web page you can convert the form into a HTTP request (POST or GET) and send it direct to PureResponse at the following address:

http://response.pure360.com/interface/list.php

Note: `response.pure360.com` in the address above can be replaced with your domain mask.

Here are a couple of differences from the way the web page works.

Field Name	Description
successUrl	This must be set to `NO-REDIRECT` to prevent PureResponse returning a redirection request. If everything works OK then you’ll received a response of `OK`.
errorUrl	You don’t need an error URL. If an error occurs then you’ll get a response of: `ERROR:` *specific error message*.

3.2 Opt-out a record

When an email address or mobile number is opted out it will no longer be included in deliveries from PureResponse. Most recipients opt-out by using the link at the bottom of the email that they receive. However, you may want to provide other ways to do it, and this interface can help.

This opt-out interface is much the same as the sign-up page and direct request above, so follow the process described above first. Once you have a working sign-up page then you can make these amendments to change it to an opt-out request.

Field Name	Description
mode	You'll need to add this field to the page or request and set the value to `OPTOUT`. `mode` is case-sensitive.
Custom Fields	You don't need to pass any custom data in this request, just make sure that you pass either the email address or mobile number.

3.3 Advanced options

We know that every client is different and so we've provided some additional options which can change the way the sign-up page process will work. It's very important that you speak to your account manager before implementing these options.

- **Removing double opt-in**

We strongly advise you to use a double opt-in mechanism for your list sign-up process; it's the only way to ensure that your new recipient really did want your email. However, it's not a legal requirement in the UK and you can remove this option if you want to. If you do, it will mean that any email address received from the sign-up form can be sent to immediately, but you haven't confirmed that the person signing-up is definitely the owner of the email address.

Just add the following HTML code into your sign-up form to switch it off:

```
<input type="hidden" name="doubleOptin" value="false" />
```

- **Replace the default double opt-in message with your own message**

You can create your opt-in email within your profile. Don't forget to include the following 'opt-in' link otherwise your recipients won't be able to complete their double opt-in process. The link is:

```
http://{~customDomain~}/ act/optin.php?id={~mailId~}
```

Once the above has been correctly configured, the sign-up creation process within PureResponse will guide you through setting up the sign-up form with the alternative opt-in message.

- **Direct opt-out**

You may choose to use a direct opt-out process to allow recipients to unsubscribe themselves without having to confirm again on the landing page.

To achieve this you'll need to use following URL as the unsubscribe link:

```
http://{~customDomain~}/ act/optout.php?id={~mailid~}
```

Any recipients that click on this link will automatically unsubscribe themselves from future campaigns without the need to confirm their opt-out choice on the standard landing page.

4 List Sign-Up Preload

If you've created a sign-up page to allow a recipient to join your list, then you can use this interface to pre-populate that page with the recipient's data. They can then edit their data and update PureResponse. This is especially useful for adding an "edit my preferences" type link to your emails.

4.1 Editing records from web pages

We'll use an example to help. In this example, the customer has set-up a sign-up page by following the explanation above. Its sign-up page contains the following fields and is hosted at http://www.xyz.com/signup.html:

- Email
- First name
- Surname
- Favourite type of film
- Favourite type of food

The customer has been using this sign-up page for a while but has noticed that its recipients are emailing to ask to change their preferences. For each one the customer receives, it logs into PureResponse, finds the recipient record and edits the custom data.

Eventually the customer decides that it would be much better if there was a link in their email saying "edit my preferences" that went to a page where the recipient could change the preferences themselves. The customer doesn't have its own development team and so wants to use PureResponse to do it.

Here is the URL that the customer links the "edit my preferences" text to in their email:

```
http://response.pure360.com/interface/signup_preload.php?url=http://www.xyz.com/signup.html&email={~email~}
```

Notice that the URL contains the full address of the sign-up page along with a personalisation field for the recipient's email address.

When the next delivery is sent out, this URL is populated with the email address of the recipient.

When the recipient clicks on the link they are redirected to the sign-up page, but this time it is pre-populated with their existing data. They can then change their data and submit, and their record in PureResponse will be updated.

If the recipient changes their email address, then a new record will be created and the old record left in place. The double opt-in procedure will then take place for the new email address.

Customers using masked domains can replace **`response.pure360.com`** with their masked domain.

5 Automated List Upload

This interface allows you to directly link your system to PureResponse and automatically create, replace, and append to, lists within your account.

Please use the information in this section in combination with the HTML example files included in the pack:

...API Developer Pack\Automated List Upload\Head_list_upload_meta.html

...API Developer Pack\Automated List Upload\Head_list_upload_data.html

5.1 Overview

You use this interface to automate the process to occur when you click a button in your own system, at a chosen times of the day or maybe on another event-based trigger.

However you choose to implement it, the interface is the same. It will require access to programming resources at your end, so if you don't have access to these then you may want to speak with your account manager to discuss alternatives.

5.2 What can be done

Using this interface will allow you to perform the following actions with lists:-

- **Create**

This action creates a new list within your PureResponse profile. The new list will be available within PureResponse with the name, language, and data as specified via the interface. This may be appropriate where you're using your CRM to filter data and want that data available within PureResponse.

- **Append**

This action adds additional records to or updates existing records in a list within your PureResponse profile. The data must match the requirements of the existing list ie. same custom fields etc.

Records containing new email or mobile numbers will add a new record to the list.

Where the email address or mobile number exists in the uploaded list and also in the exiting list on PureResponse, here is how the system behaves:

For each field in that record that contains data that data will overwrite whatever is in the field in the existing list.

For each field in that record that contains no data, that field in the existing list will remain unaltered.

Note: Is is not possible to APPEND records to a list in such a way as to remove data which exists in a field and result in that field in the existing list being left empty.

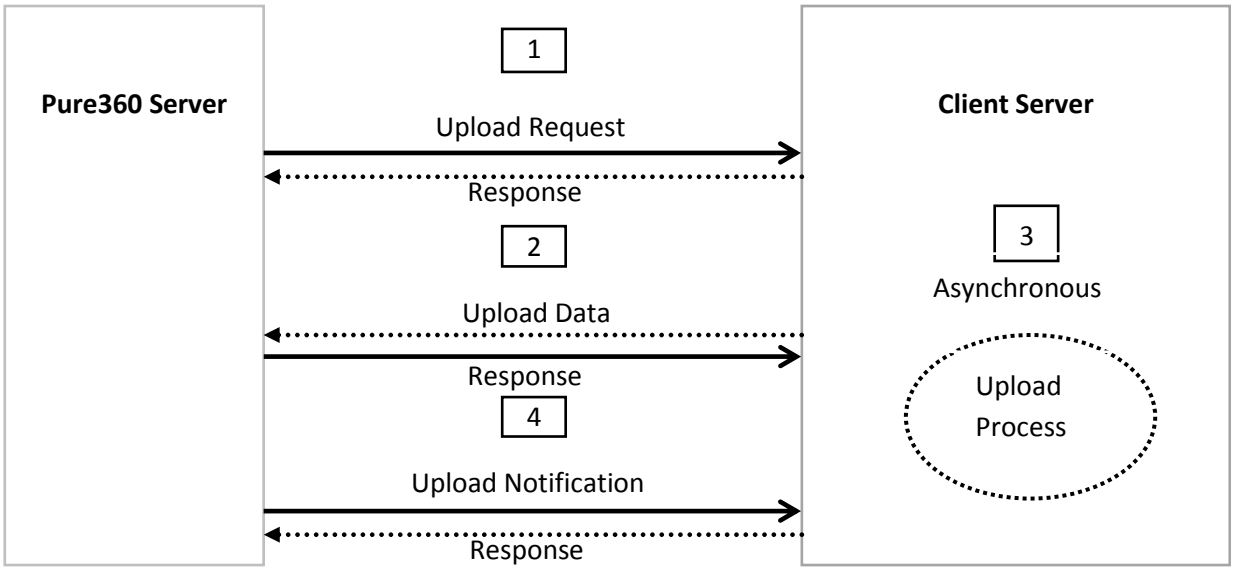
- **Replace**

This action replaces an existing list within your PureResponse profile. The list is deleted and the new list put in its place. This may be appropriate where the client regularly sends to a list based on the same criteria within their CRM, and wants the latest data available.

5.3 How it works

This interface works in an asynchronous mode and the conversation between your system and PureResponse is made up of more than one communication. The diagram below shows the four parts of the process. All the functions described above require the same communication and are based upon sending a request that explains the data followed by the data itself.

List Upload: Server-to-server procedure



We will now describe each of the four parts of this process in more detail. Note that the three different types of request (i.e. create, append and replace) all work in the same way.

5.4 Part 1: Upload (meta) request

The upload request is an HTTP post from your system to PureResponse. It is a request for a transaction id for your upload and carries details of what PureResponse can expect within the data.

The upload request message contains all of the information required to create a list record on the system with the exception of the list data itself.

Here are the details of this request:

`http://response.pure360.com/interface/list_upload_meta.php`

(Alternatively, you can use your masked domain).

Parameter Name	Parameter Description
profileName	Set this to the name of your profile. The profile name can be found at the top of the screen when logged into PureResponse.
listName	Set this to either the name of the list being created or the existing list on the account. Make sure you keep the same case and any spaces etc to avoid problems.
languageCode	Set this to the language code eg. `en_GB.ISO8859-15` for the list. If not supplied the list will be inserted with the default language code for the account.
transactionType	This determines whether the list should be created `CREATE`, appended to an existing list `APPEND`, or replace an existing list `REPLACE`.
emailCol	The number of the column within the data file that contains the email (if applicable).
mobileCol	The number of the column within the data file that contains the mobile number (if applicable).
COL_xxxx	The number of the column within the data file that contains the custom field. Where `xxxx` is any string that would be a valid field name on the system. A maximum of 10, 20 or 40 custom fields can be defined, depending on your package. If more are defined then the first ten only will be used.
responseType	Either `EMAIL`, `HTTP` or `REST`
responseUri	Either the email address *or* response URL to send the HTTP or REST upload notification to.

Note: The first column in the data file is numbered 0 (zero), so if the email address is in the first column you should pass an emailCol value of '0', if in the next column you pass '1' etc.

As an example, imagine that you want to upload the following data in a list called `My List`, in the `myProfile` profile, and set the language to German:

```
john.smith@email.com,John Smith,07798564352,Brighton
sarah.jones@email.com,Sarah Jones,0779646352,London
bob.samuel@email.com,Bob Samuel,0775354542,Cardiff
```

Your request would contain the following values:

Field	Data
profileName	myProfile
listName	My List
languageCode	de_DE.ISO8859-15
transactionType	APPEND
emailCol	0
mobileCol	2
COL_fullname	1
COL_town	3
responseType	HTTP
responseUri	http://www.xyz.com/reply.jsp

- **Validation**

We have to make sure that everything you pass to us is correct. So that we can save you some time, here are the main validation rules that we will be applying:-

Name	Description
profileName	This must match an existing profile name and you must have interfaces enabled. If you're not sure what this is you should speak to your account manager.
listName	The validation depends upon the value you pass in the `transactionType` column as follows:- `CREATE`: The list name must not match an existing list name for the profile you've specified. `APPEND`: If the list name does not already exist for the profile specified then this is treated as a `CREATE`. If the list name does already exist then the list you've referred to must not be part of a pending delivery. `REPLACE`: The list name must match an existing list name already loaded into the profile you've specified, and the list must not be part of a pending delivery.
languageCode	The language code must be a supported ISO language code and must be a language available to the specified profile. Example supported language codes:

'	en_GB.ISO8859-15 – English de_DE.ISO8859-15 – German fr_FR.ISO8859-15 – French Language code is ignored for APPEND transactions unless the specified list does not already exist.
transactionType	CREATE, APPEND or REPLACE only.
emailCol	This must be a positive whole number. You must provide either an emailCol or mobileCol or both.
mobileCol	This must be a positive whole number. You must provide either an emailCol or mobileCol or both.
COL_xxxx	This must be a positive whole number if supplied. If the transactionType is APPEND and the list name already exists for the profile specified, then all fields already defined on the existing list must be supplied as COL_ fields here. **Note**: If the transactionType is REPLACE then the COL_ fields do not have to match the existing list.
responseType	Must either be EMAIL, HTTP or REST.
responseUri	Must be a valid email address if responseType is set to EMAIL, or must start with http:// if response type is set to HTTP.

- **Response**

The system will validate the data received and return a response message containing the transaction id to be used with the data upload request. See the appendix for more information about the structure of the response messages.

Note: You need to store the transaction ID that we give you in response to the 'meta' request. This ID will need to be supplied in part 2 when you send your 'data' request.

5.5 Part 2: Upload (data) request

The upload data message is an HTTP post that contains the actual list data itself. You also send us the profile name again, and the transaction id you received from part 1. We use these to match your data with the column definitions you sent us.

Here are the details of this request:

Parameter Name	Parameter Description
profileName	Same as before.
file	You need to pass us the file data within a multiple-part form post. We've included some extra information about this in the appendix. **Note**: You may if you wish replace the `file` parameter name with the transaction ID value from part 1 (the 'meta' request) as the parameter name. Doing this removes the requirement to use the `transactionId` parameter separately. The file data itself must be in CSV format. Each record separated by a new line, and each field separated by a comma. Commas are not required at the start and end of each row. The order of the data within each row must be the same, and must match with the column definitions provided in the transaction request. For performance, the text files can be compressed using ZIP archives, GZIP or BZIP2. These archives must contain a single CSV file as described above, and you'll need to change the content type to reflect the new data being passed.
transactionId	The transaction ID received from part 1, the 'meta' request.

5.6 Part 3: Asynchronous upload

This part is completed by PureResponse, and because it is asynchronous, your system can be getting on with other things while this happens.

Once a valid list data upload has been completed, the list is added to a queue internal to PureResponse. At regular intervals, PureResponse will collect and process the lists. Larger lists take longer to process and the length of time will also depend on the number of lists on the queue.

Once the list is uploaded it will be available in the list view within PureResponse.

5.7 Part 4: Upload Notification

You will probably want to know when the list upload process is complete in order that your system can perform some further actions etc. We send either a web request (POST) to your system at the URL that you gave us in part one, or send an email to the email address you gave us in the same way that PureResponse does normally.

- **HTTP response**

If the list uploaded successfully, then the body of the request (POST - content type*: application/x-www-form-urlencoded*) will contain `OK:` followed by the transaction ID so that you can identify the original request. If an error occurred then the body will contain `ERROR:` and the reason why. See the appendix for the standard response message format.

Finally, we want to make sure that you received our response okay, so we need you to return an `OK` response confirming that the notification has been received. The system attempts to send the message again an hour later hour if no response is received. The system notifies us if this continues.

- **REST responses**

If the list uploaded successfully, then the web request (POST - content type*: multipart/form-data*) will contain the following:

`[identityName] =>` *profile name*
`[listName] =>` *list name*
`[rowsUploaded] =>` *count of rows uploaded*
`[duplicates] =>` *count of duplicate rows*
`[failedUploadUrl] =>` *link to a csv containing failed records*
`[supportEmail] =>` *support email for this profile*

In the same way as the `HTTP` response type, we want to make sure that you received our response okay, so we need you to return an `OK` response confirming that the notification has been received. The system attempts to send the message again an hour later hour if no response is received. The system notifies us if this continues.

6 Post-Click Tracking

This interface allows you to record what happens on your website after the recipient has clicked on a link in your email, and then see the results inside PureResponse.

We understand that not everyone has a fully fledged tracking system running behind their website. However, we don't think that this should stop you knowing more about your recipients and what happens when they leave your email and move around your website.

Post-click tracking allows you to identify the key pages you want to monitor on your site, and keep track of who has visited these pages from links in your emails. Not just the page they go to first, but also the next page, and the next etc. You can even record a little extra information such as how much they spent, or how many items they bought.

6.1 How it works

Post-click tracking is achieved by PureResponse placing a cookie in the browser when a tracked link is clicked in your email. If your recipient keeps their browser open and reaches one of your tracked pages, then your website tells PureResponse to track the event, and passes with it the cookie that identifies who it is.

The tracked data can be viewed within the campaign detail report via an additional section that will only appear once at least one item has been tracked.

6.2 Setting up

Setting up requires the following two stages:-

- **Creating a special tracked link**

The link from the email to the first web page must contain an indicator to instruct PureResponse to deposit the cookie in the memory of the web browser as follows:-

Normal link:-

`http://www.foo.com/salespage.html`

Sales Tracking Link:-

`http://www.foo.com/salespage.html?_pureTrackingName=FOOSALES`

In this example we have chosen to use the word `FOOSALES` to uniquely identify the tracked process. You should choose an appropriate name to use that will identify you and your tracked process. This prevents problems with recipients that receive two tracked emails from PureResponse and click on the links within the same web browser session.

PureResponse will obscure the tracking attribute in the URL so that your recipient won't see `_pureTrackingName` in their browser.

Don't forget that the email must be set to track links.

- **Creating a special tracked page**

We need your server to send PureResponse a notification when the event is ready to be tracked. This is done by placing an (invisible) image on the page.

The following example shows an image tag to track the event. This tag may be placed on any page you want to track (for example):-

```
<IMG
SRC="http://response.pure360.com/_act/tracking.php?_pureTrackingName=FOOSAL
ES&type=SALE&desc=49.99" />
```

In this example we have matched the tracking name of `FOOSALES`. It is crucial that this matches the name in the link in the email.

The type `SALE` will appear on the reports and can be set to anything you choose but it has to be set to something. If you want to identify each different page then give each page a different “type" value.

The description (`desc`) attribute is optional and you may like to set this dynamically so that it contains extra information e.g. how much was bought.

IMPORTANT: If the profile within PureResponse is using web masking then the URL in the image `SRC` attribute above must replace the `response.pure360.com` domain with the custom web domain.

6.3 Secure websites

We’re often asked if this will work on secure websites. Unfortunately (or fortunately), browsers don’t like secure websites to be pointing to different locations or a mix of secure and insecure locations. This means that if you want to do this on a secure website then we recommend that you use a direct server call. This will require access to some programming skills so it may not be so useful to you, but you can achieve it as follows:-

Step 1. Add `mailId={~mailId~}` to the end of the links in your message that lead to your website.

Step 2. Get your programmer to capture the value passed in for `mailId` and store it during the recipient’s session.

Step 3. When your server wants to send a post-click tracking event then get it to submit the following URL:-

```
http://response.pure360.com/_act/tracking.php?id=#id#&type=SALE&desc=49.99
```

Where you replace the `#id#` with the id you’ve been storing, and the other parameters with your own values.

7 PAINT

The PureResponse API is run on a software layer within our system we call **PAINT** (**P**ureResponse **A**pplication **INT**erface).

PAINT makes available to you (via a SOAP service) many of the functions that are available within PureResponse e.g. creating messages, uploading lists, scheduling deliveries, and accessing reports.

7.1 Overview

The PAINT specification is intended to meet your custom requirements for integration between parts of our email and SMS marketing system and your own systems and websites.

PAINT separates the business logic and processing of the PureResponse system from the online PureResponse user interface. It is because of this separation that you can include functions, such as generating new emails or scheduling a delivery, within your own systems.

7.2 Using PAINT

PAINT can be used in the two following ways:-

SOAP – This is a standard method of communicating between two systems on the web. It is available to most web application developers.

PAINT Wrappers – These are wrappers that perform the specific jobs whilst hiding the technical details of using SOAP.

Which method to choose?

You could use either or both methods to access PAINT.

Here is a guide to choose which method is best for you:

If the answer to any of the following is *yes*, then the SOAP solution would be best:

- You need maximum flexibility to integrate into our systems.
- You have development resources that can use SOAP.

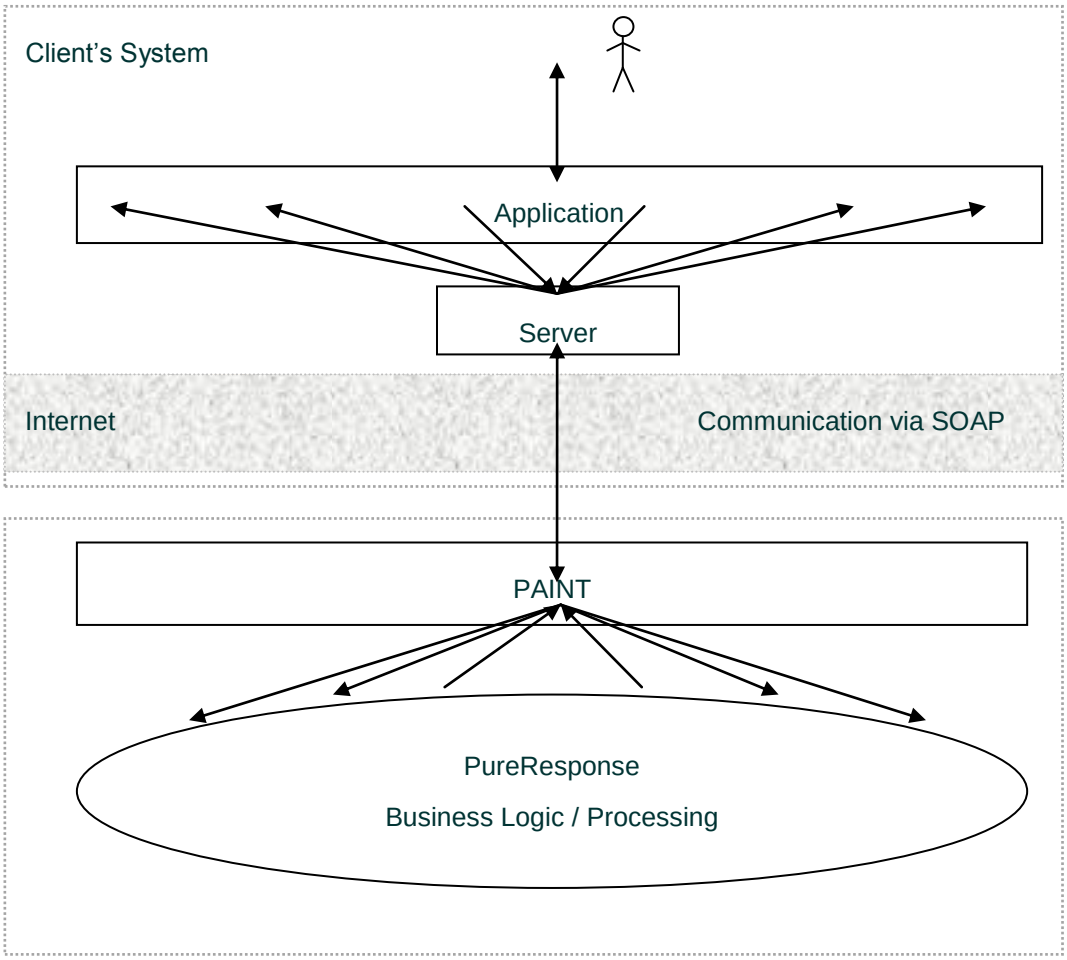
If you have answered *no* to any of the questions above, then you could consider using PAINT wrappers built by Pure360 to your specification.

The SOAP method is the most flexible and powerful, so we'll describe it here in this section and then in later sections we'll describe the other method.

7.3 SOAP

SOAP (Simple Object Access Protocol) is a standard way for two systems to share functionality with each other. Once the functionality has been set-up in this way, one server can access functionality running on the other server as if it were running locally.

The following diagram shows how SOAP allows your server to access PAINT functionality invisibly to the user.



Here are some key features:-

- SOAP is built into most web development tools e.g. Java, PHP and .NET, thus reducing development time.
- SOAP provides the maximum flexibility when choosing how to use PAINT.
- Your server logs in and is therefore able to use PAINT's data caching facilities.
- Secure web connections (HTTPS) are supported.

SOAP is used by your own application servers and so can be plugged into any form of application e.g. website, Intranet, Windows application etc. Appendix B has more detailed information about how to use SOAP to connect to PAINT.

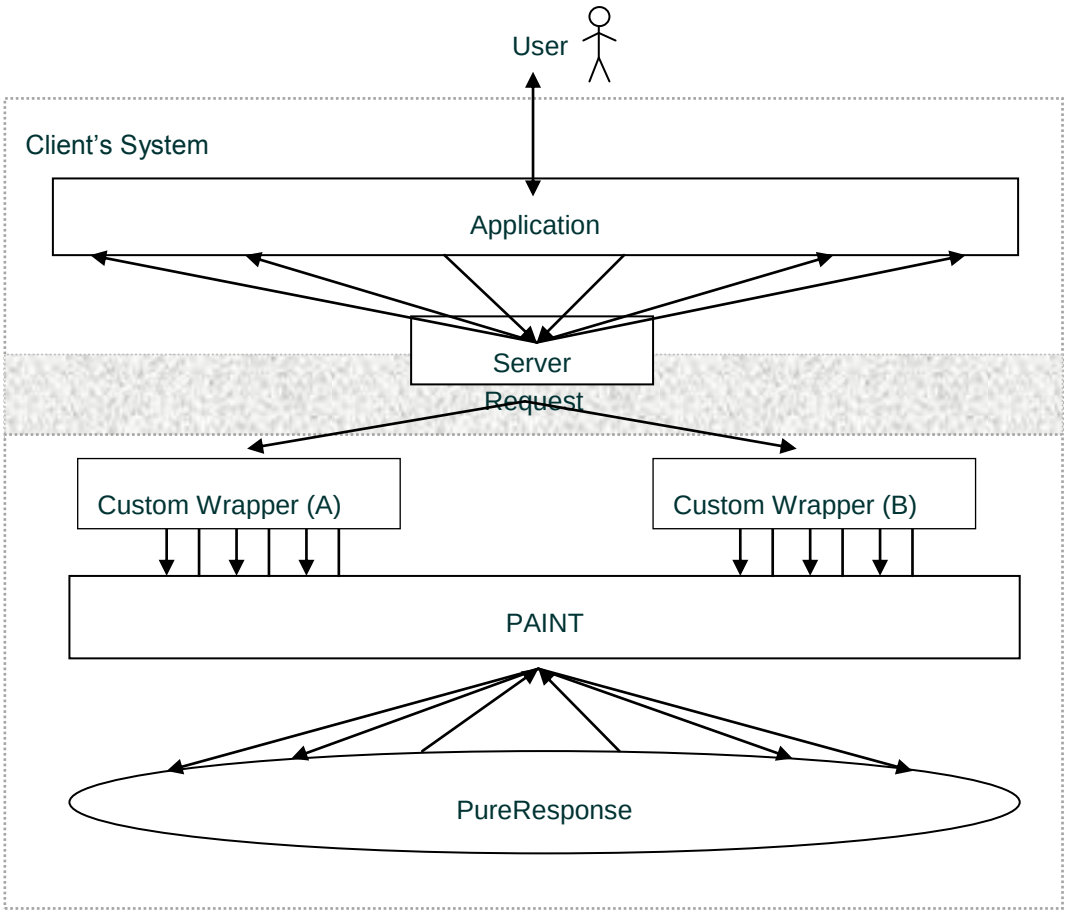
8 PAINT Wrappers

If you don't have the time or resources to develop using SOAP, then we can create 'wrappers' that perform the specific set of functions you require but using a simple HTTP web request.

If you don't have access to SOAP development resources then you might like to try the option of using PAINT wrappers.

PAINT wrappers are bespoke modules that encapsulate pre-defined sets of functionality within web requests. An example of such a wrapper may be the request to create a new email based on a template name and a plain text message. The PAINT wrapper does the necessary logging in, email creation, update, and logging out, without the need for SOAP calls by your servers.

The following diagram shows how PAINT wrappers allow your server to access a predetermined collection of PAINT functionality invisibly to the user.



Here are some key features:-

- No requirement for SOAP development resources. Called via a web request.
- Less flexible than using SOAP directly because the functionality of custom modules is predefined and then fixed.
- There are fewer options for returning data back to the calling server. As the sophistication of the data returned increases, the cost effectiveness of PAINT wrappers will reduce.
- No state is maintained in PAINT between each call to a PAINT wrapper, so the data caching facilities within PAINT are not available.
- Secure web connections (HTTPS) are supported.

An example of one wrapper is provided within the API developer pack. Its function is to trigger a One to One message from PureResponse:

```
...API Developer Pack\PAINT Wrapper\One2one\oneToOneForm.htm
```

9 Event Notifications (Outbound)

These notifications can be sent in near real time via email, over the web to an application, or to automatically update a CRM system, or can be held by us until you ask for them. They can be sent when a single recipient bounces (hard bounce / blocked / soft bounce), opts in, opts out, opens or clicks a link. They can also be sent when your deliveries start and finish.

Your data is vital for your business, and keeping the central store of data (held in your systems) up to date can only be a good thing. Unfortunately, there isn't a global standard for feeding back email marketing information into every database that is being used, and so we've had to make our system flexible so it can cater for a wide variety of requirements.

We'll first explain the different ways that PureResponse can send you these notifications. Once we've done that, we'll explain what events we notify you about and tell you what information you'll receive in each case.

9.1 How we send you notifications

There are currently three ways that we send you notifications, but we may add more in the future. You'll need to choose one of these methods.

- **Email**

The simplest option is that we can send an email, to an address specified by you, each time we have a new notification. Of course, if you're sending out very large numbers of emails, you won't want this address to be a person, but maybe you'd like a computer system to receive these notification emails.

- **Web request**

We can send a web request (POST) each time an event occurs. That way you can set-up your system so that it automatically updates your data whenever PureResponse tells it about an event.

To set-up a web request, you need to provide us with a URL for these events. Alternatively you can group event types to a URL or specify a separate URL for each different type of event. We will then send a request to that URL when the event occurs. The keys and values we send are described later in this section.

You'll need to write the code that can receive these web requests and process the data accordingly.

Notes: We do expect a success response message i.e. an "OK" response reply that needs to come from your own server. If we don't get that reply, our servers will attempt to send the data up to five times before giving up, so you may receive quite a few duplicates if this occurs.

Note: HTTP post is not recommended for any user that is sending 500k emails or more per week. Please contact your account manager to discuss this further.

- **Pull**

For some customers, the idea of having lots of data passed in at different times of day is unacceptable and a batch option is preferable.

We provided a PULL option for event notification. Asking for the data is done through PAINT, so you'll need to get your PAINT interface working and then work through the platform example implementation to see how you retrieve the data we store.

9.2 Events we notify you about

Each of the following events can be either enabled or not, it's up to you. Here is an explanation of the events and the different data that is received based upon the way you've asked us to notify you.

- **Open**

We can send you a notification when someone opens their email. In order to have this on your account you'll need to contact us to set it up. Once it is enabled, then you can drop the following image into any of your messages and it will result in an open notification being sent to you.

```
<IMG
SRC="http://response.pure360.com/_act/tracking.php?triggerType=OPEN&id={~mailId~}" />
```

The `{~mailId~}` field will be automatically converted to the unique reference for the recipient, and this allows us to send you the correct information. Note that you can use your masked domain instead of response.pure360.com if you have one set-up.

Here is the data that you will receive based upon the type of notification you've chosen:-

Notification Type	Data Received
Email	Subject: OPEN notification: #email address# Plain text body only: #email address#
HTTP Post	**[type]** => OPEN **[Event_date]** => Event datetime unix timestamp **[email]** => Recipient email address **[mobile]** => Recipient mobile number **[messageName]** => Campaign message name **[deliveryDate]** => Delivery date unix timestamp **[device_type]** => Type of device used to open the message **[device_OS]** => Operating system of device used to open the message **[device_Browser]** => Browser of device used to open the message **[Delivery_ID]** => Delivery ID **[*custom data field(s)*]** => list custom data field(s) values **[notificationSubject]** => Shown if subject line selector used
PULL request	**[type]** OPEN **[email]** Recipient email address **[messageName]** Campaign message name **[deliveryDate]** Delivery date unix timestamp **[*custom data field(s)*]** list custom data field(s) values

- **Click**

A click notification can be applied to any tracked link in your message. You must first contact us to get the click trigger enabled, then after that has been done, you'll need to do the following to activate each individual link:-

Choose the link or links that you want to receive notifications for. Then add the special identifier to the end of the link as a valid URL parameter. Your programmer will be able to help with this.

Normal link:-

```
http://www.foo.com/salespage.html
```

Click trigger link with special identifier:-

```
http://www.foo.com/salespage.html?_pureTrigger=on
```

This addition has enabled click notifications for this link. Don't forget that you'll need to switch on tracked links for this to work and it will only work in a full delivery.

PureResponse will obscure the trigger attribute in the URL so that your recipient won't see `_pureTrigger` in their browser.

If the trigger is set-up to issue an email rather than a web request, then it is possible to specify an alternative email address for each link too. The following example shows a link that uses the click trigger, but that sends the email to the address `alternative@foo.com`:

```
http://www.foo.com/salespage.html?_pureTrigger=on&_pureTriggerEmail=alternative@foo.com
```

Again, the `_pureTriggerEmail` parameter will be obscured before it's shown in the recipient's browser.

Here is the data that you will receive based upon the type of notification you've chosen:

Notification Type	Data Received
Email	Subject: CLICK notification: #email address# HTML and plain text body: #data as per web request below#
HTTP Post	**[type]** => CLICK **[device_type]** => Type of device used to open the message **[device_OS]** => Operating system of device used to open the message **[device_Browser]** => Browser of device used to open the message **[email]** => Recipient email address **[deliveryDate]** => Delivery date unix timestamp **[Delivery_ID]** => Delivery ID **[messageName]** => Campaign message name **[*custom data field(s)*]...** => list custom data field(s) values... **[notificationSubject]** => Shown if subject line selector used **[clickUrl]** => URL of clicked link **[clickDate]** => Click datetime unix timestamp **[mailVariationId]** => Split content ID
Pull requests	**[type]** CLICK **[email]** Recipient email address **[Message Name]** Campaign message name **[Delivery Date]** Delivery date unix timestamp **[Click Url]** URL of clicked link **[Click Date]** Click datetime unix timestamp **[*custom data field(s)*]** list custom data field(s) values

- **Bounce, blocked, soft bounce, opt-in and opt-out**

We've put these together because they are so similar, but you can enable them independently. All you need to do to set-up one of these notifications is to contact us. Once it is enabled, you'll automatically start receiving updates when they occur.

> If you're setting up bounce notifications, then we recommend that you do your first full send without this enabled. First sends tend to result in more bounces, and you may not want to overload your systems or your inbox with lots of data.

Here is the data that you will receive based upon the type of notification you've chosen:-

Notification Type	Data Received
Email	Subject: BOUNCE/BLOCKED/SOFTBOUNCE/OPTIN/OPTOUT notification: #email address# Plain text body only: #email address#
HTTP Post	**[type]** => BOUNCE/BLOCKED/SOFTBOUNCE/OPTIN/OPTOUT **[Event date]** => Event datetime unix timestamp **[email]** => Recipient email address **[messageName]** => Campaign message name **[deliveryDate]** => Delivery date unix timestamp **[device type]** => Type of device used to open the message **[device OS]** => Operating system of device used to open the message **[device Browser]** => Browser of device used to open the message **[Delivery ID]** => Delivery ID **[custom data field name]** => custom data field value
Pull request	**[type]** BOUNCE/BLOCKED/SOFTBOUNCE/OPTIN/OPTOUT **[email]** Recipient email address **[messageName]** Campaign message name **[deliveryDate]** Delivery date unix timestamp **[*custom data field(s)*]** list custom data field(s) values

Note: Optin events can also occur as the result of a list signup. This means that it is possible to receive these notifications without a messageName or a deliveryDate parameter being supplied.

- **Delivery Started**

This option sends a notification when the delivery has started processing. It is useful if you want to be notified about when a delivery starts and to which lists the message will be sent to.

As with the above, you just need to contact us to set-up this notification, and it will be automatic from then on.

Here is the data that you will receive based upon the type of notification you've chosen:-

Notification Type	Data Received
Email	Subject: DELIVERY notification: #message name# Plain text body only: #email address#
HTTP Post	**[type]** => DELIVERYSTART **[Profile]** => Profile name **[Message_Type]** => EMAIL **[messageName]** => Campaign message name **[Delivery_ID]** => Delivery ID **[List_Names]** => Comma separated list names **[Ext_List_ID]** => External list key if imported from an external system **[Delivery_Scheduled_Date]** => Delivery scheduled date unix timestamp **[Delivery_Status]** => Current delivery status (normally PROCESSING) **[Delivery_Start_Date]** => Delivery start date unix timestamp **[Delivery_Complete_Date]** => *[no value]*
PULL request (default data sent with each request)	**[type]** DELIVERYSTART **[subject]** *[no value]* **[User]** *[no value]* **[Delivery Start Date]** Delivery start date unix timestamp **[Delivery Complete Date]** *[no value]* **[Message Name]** Campaign message name **[List Names]** Comma separated list names **[Emails Sent / Credits Used]** *[no value]*

Note: Delivery notifications are occasionally suffixed with an email address. This is taken from the contact record of the person scheduling the delivery. If the delivery occurred as part of a recurring delivery, this parameter will not be available.

- **Delivery Completed**

This final option sends a notification when the delivery has completed processing. It is useful if you want to be notified about who is creating deliveries and how many are being sent.

As with the above, you just need to contact us to set-up this notification, and it will be automatic from then on.

Here is the data that you will receive based upon the type of notification you've chosen:-

Notification Type	Data Received
Email	Subject: DELIVERY notification: #message name# Plain text body only: #email address#
HTTP Post	**[type]** => DELIVERY **[notificationSubject]** => Message name **[User]** => Login used to schedule delivery **[Delivery_ID]** => Delivery ID **[Profile]** => Profile name **[Message_Type]** => EMAIL **[Delivery_Scheduled_Date]** => Delivery scheduled date unix timestamp **[Delivery_Start_Date]** => Delivery start date unix timestamp **[Delivery_Status]** => COMPLETED **[Delivery_Complete_Date]** => Delivery completion date unix timestamp **[Message_Name]** => Campaign message name **[List_Names]** => Comma separated list names **[Emails_Sent_/_Credits_Used]** => Emails Sent / Credits Used **[_interfaceSendEmail]** => Override email address for event notifications
PULL request	**[type]** DELIVERY **[subject]** *[no value]* **[User]** Login used to schedule delivery **[Delivery Start Date]** Delivery start date unix timestamp **[Delivery Complete Date]** Delivery completion date unix timestamp **[Message Name]** Campaign message name **[List Names]** Comma separated list names **[Emails Sent / Credits Used]** Emails Sent / Credits Used

Note: Delivery notifications are occasionally suffixed with an email address. This is taken from the contact record of the person scheduling the delivery. If the delivery occurred as part of a recurring delivery, this parameter will not be available.

- **SMS Opt-out, SMS Failed**

We've put these together because they are so similar, but you can enable them independently. All you need to do to set-up one of these notifications is to contact us. Once it is enabled, you'll automatically start receiving updates when they occur.

> Remember that on your first send, you are likely to receive a high number of SMSOPTOUT and SMSFAILED notifications. Ensure that your systems are in a situation to be processing the potentially high number of these notifications.

Here is the data that you will receive based upon the type of notification you've chosen:-

Notification Type	Data Received
Email	Subject: SMSOPTOUT/SMSFAILED notification: #mobile number# Plain text body only: #mobile number#
HTTP Post	**[type]** SMSOPTOUT/SMSFAILED **[Event date]** => Event datetime unix timestamp **[mobile]** Recipient mobile number **[messageName]** Campaign message name **[deliveryDate]** Delivery date unix timestamp **[Delivery ID]** => Delivery ID **[custom data field name]** => custom data field value
Pull request	**[type]** SMSOPTOUT/SMSFAILED **[Event date]** => Event datetime unix timestamp **[mobile]** Recipient mobile number **[messageName]** Campaign message name **[deliveryDate]** Delivery date unix timestamp **[Delivery ID]** => Delivery ID ***[custom data field(s)]*** list custom data field(s) values

10 SMS Reply Module (Outbound)

You can receive an email, SMS or web notification when an SMS is received to your keyword and shortcode. We call the product SMS Interact or SMS Interact Advanced (depending on the requirements) and it uses the SMS reply module.

If you want to market by giving out an SMS number and receiving replies back then you'll want our SMS reply module. This module can be customised specifically for you to do almost anything you'd like. What's relevant to this document is that you can request that the module acts as an outbound interface, notifying you or your systems when replies are received.

Since each module is custom made to your requirements, there are few limitations on what can be done as long as you want the response sent over the Internet.

Here are some examples that may help you decide on your requirement:-

- Send an SMS to a specified address.
- Send an email to a specified address.
- Send a web request to your system.
- Send XML over HTTP to a web service.

Whatever solution you require, your next step should be to contact your account manager and discuss your requirement.

11 Appendix A – Automated List Upload

11.1 Response message format

There are two types of response messages: success and error.

The success response message will be in the format `OK: #additional info#` and the error message will be in the format `ERROR: #additional info#.`

`#additional info#` is any additional information, e.g. transaction id or error message.

11.2 Example upload data message

Here is an example of an “upload data" HTTP post. This example would be the data upload for transaction id “123456789" for a profile named “myProfile"; we’ve highlighted the two values in red.

```
POST /interface/list_upload_data.php HTTP/1.1
Host: response.pure360.com
User-Agent: www.xyz.com upload process
Accept: text/plain
Accept-Language: en
Accept-Encoding: gzip
Accept-Charset: ISO-8859-1
Keep-Alive: 3600
Connection: keep-alive
Referrer: http://www.xyz.com/uploadRequest.jsp
Content-Type: multipart/form-data; boundary=thisistheboundary123
Content-Length: 516

--thisistheboundary123

Content-Disposition: form-data; name="myProfile";
myProfile

--thisistheboundary123

Content-Disposition: form-data; name="123456789";
filename="listData.csv"
Content-Type: text/plain

john.smith@email.com,John Smith,07798564352,Brighton
sarah.jones@email.com,Sarah Jones,0779646352,London
bob.samuel@email.com,Bob Samuel,0775354542,Cardiff

--thisistheboundary123--
```

12 Appendix B – How to: PAINT via SOAP

In this second appendix, we'll explain some of the more technical detail of how PAINT is accessed via SOAP. We'd strongly recommend that you only read this section if you already understand SOAP and how to implement SOAP in your environment.

> Note: Within this explanation we use the word "bean". It's a bit of an odd word and has been "borrowed" from the Java concept of beans. When we use it, we mean a class that encapsulates fields and methods for a single logical entity (e.g. email, list or 'thing').

12.1 Overview

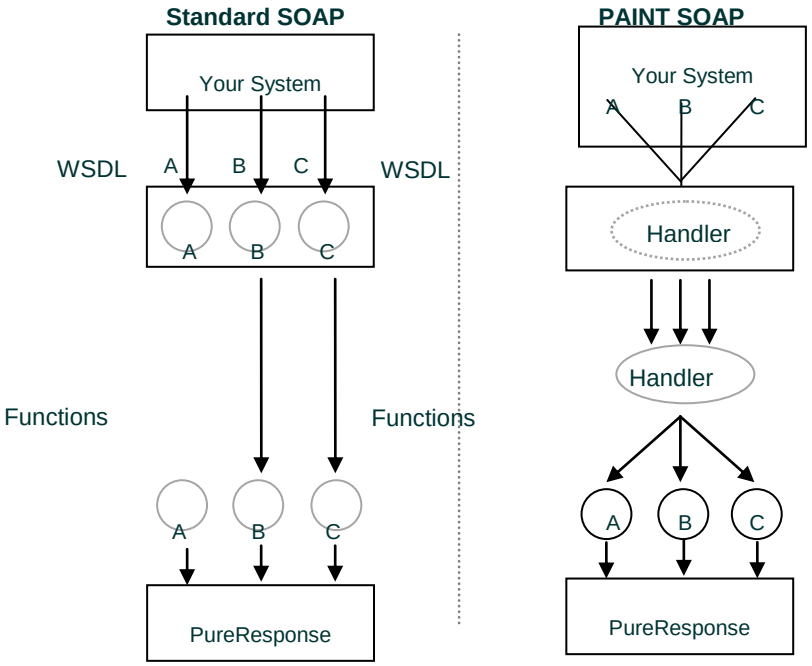
As you will know, a SOAP conversation consists of a request followed by a response. If we use an example, the request could be "get me the data about email number 123" and the response would be "here is the data for email 123".

We'll start by explaining the SOAP format that we're using. We'll then explain the request and response conversations in more detail followed by information about tying several of these conversations together over time into a "session". Once the mechanics of the conversation are completed, we'll explain how to choose what you want to 'say' (do) within PAINT.

12.2 The SOAP format

A single function

SOAP is designed to describe lots of different functions within a system and make them available to you as if they were running within your own system. We use it slightly differently to this because we have defined one generic function, and we use that function to invoke everything by passing it additional parameters.



You may not agree with this approach, but we've done it to simplify the interface, to maximise flexibility and to work more smoothly with the other uses of PAINT, so we hope that you won't find it too abhorrent.

WSDL files

We have chosen to declare our SOAP function using a WSDL (Web Service Definition Language) file.

For clients with non-unlimited systems, we have implemented two WSDL files; one for use with RPC/encoded and the other for use with RPC/Literal. We recommend that you use the RPC/literal file unless developing in PHP5. The files are located at the following addresses, and you should be able to read the files within your web browser.

RPC/encoded:
`http://paint.pure360.com/paint.pure360.com/ctrlPaint.wsdl`

RPC/literal:
`http://paint.pure360.com/paint.pure360.com/ctrlPaintLiteral.wsdl`

For clients with unlimited systems, we have created a white labelled, unbranded version of the two WSDL files; This URL can be used in place of the original urls and although it still points to Pure360, it contains no references to Pure360.

Note: This should be of use to clients who wish to build third party solutions using Paint but who do not wish to reference 'Pure360' within their applications.

RPC/encoded:
`http://emailapi.co.uk/emailapi.co.uk/ctrlPaint.wsdl`

RPC/literal:
`http://emailapi.co.uk/emailapi.co.uk/ctrlPaintLiteral.wsdl`

The WSDL files contain the name of the generic function, the fields being passed in, and the response data being returned. They are fully commented and explain each parameter being passed. We haven't restated it here because it's better for you to read the live files to make sure that you get the latest version.

If you are using the literal version of the WSDL then you will send and receive your data in sets of PAINT "pairs". These are described in the WSDL document. If you are using the encoded version in PHP5 then you'll send and receive associative arrays which are not defined within the WSDL but are native to PHP.

Sending a PAINT Request

You'll need to do some work to structure your SOAP request for PAINT. Fortunately, most modern IDEs will generate stub files based upon a WSDL URL. Once you have created your code to make the SOAP request you'll need to assemble your parameters.

The first parameter is the context ID. This field is passed in all requests other than the login request and ties your session together. You can find information on obtaining the context ID in the following sections on sessions and logging in.

The next two parameters describe the type of process you want to invoke. Details about the values of these parameters are provided in the step-by-step guide to selecting a process below.

The final two parameters will be used to pass the data you want to send to the process. If you are using RPC/literal you will need to ensure that you send your data in the format described by the WSDL file, however if you are using PHP5 and RPC/encoded you may pass this data in an associative array.

Please review the example implementations for a head start in your specific platform and for some utility classes for converting input and output data.

Understanding the PAINT response

When you successfully send a request to us, we will return our response containing the result of the request and any output data. The format of the response will match the format of the data passed in the request and so depends upon the WSDL file used i.e. for the RPC/literal format the data uses the WSDL definition, for RPC/encoded in PHP5 the data is returned as an associative array.

For the remainder of this document it will be assumed that “array" refers to the platform specific container you receive the data in.

This array will contain two named elements as follows:-

Element name	Type	Description
`result`	String	Type of result that occurred, either “success" or the name of the class of Exception.
`resultData`	Array	Array of data dependent upon the type of result

Here are the possible values that we’ll send you in the “result", and also what will be put into the “result Data" section based on the result.

Value	Description	Contents of result Data
`success`	The requested process Completed successfully.	Populated with nested bean data as selected by the Process called.
`bean_exception_validation`	One or more validation errors occurred.	Array of strings containing the validation error messages keyed on the fields that caused the errors.
`bean_exception_system`	A fatal system exception occurred.	String description of the error that occurred.
`bean_exception_security`	Invalid permissions for the requested action.	String description of the illegal request.

Result: success

If the result returned is “success", then the resultData element will be populated with data. The data it contains will depend upon the bean you used and the process you called.

Here is an example of a resultData array containing the data from a single eSale entity:-

```
Array

(
        [bus_entity_campaign_esale]  =>  Array
        (
[status]  =>  new
[beanId]  =>  3fb11d18e147f3e1d52e8f15c36f4c2f
[beanName]  =>  bus_entity_campaign_esale
        )
)
```

The above shows a very simple example of a response array. You can see that the first level inside “resultData" is another array element named by the type of bean data returned. Within this is the data you’ve requested related to that bean.

Why have we done it like this? Well, some processes you call will return data from more than one bean e.g. login returns context and account data. Using this structure, we’re able to return lots of different data from different beans without mixing it all together.

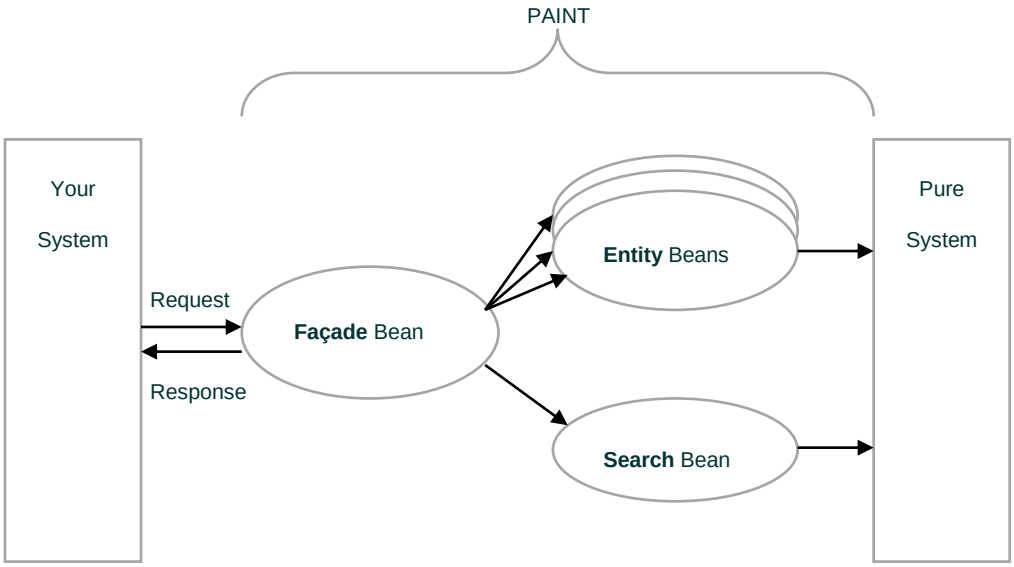
Here is an example containing data for two beans:-

```
Array
(
       [bus_entity_campaign_esale]  =>  Array
              (
                     [status]  =>  new
                     [beanId]  =>  3fb11d18e147f3e1d52e8f15c36f4c2f
                     [beanName]  =>  bus_entity_campaign_esale
              )
       [bus_entity_preferenceLogin]  =>  Array
              (
                     [loginId]  =>  123
                     [ownerType]  =>  login
                     [preference]  =>  Array
                            (
                                   [emailEdit]  =>  Array
                                          (
                                                 [displayBody]  =>  plain
                                                 [htmlEditor]  =>  wysiwyg
                                          )
                            )
                     [preferenceTypeMap]  =>  Array
                            (
                                   [emailEdit]  =>  EMAIL_EDIT
                            )
                     [preferenceNameMap]  =>  Array
                            (
                                   [displayBody]  =>  display-body
                                   [htmlEditor]  =>  html-editor
                            )
                     [status]  =>  modified
                     [beanId]  =>  busEntityPreferenceLogin
                     [beanName]  =>  bus_entity_preferenceLoginKey
                     [fieldAccess]  =>  Array
                            (
                                   [beanId]  =>  readwrite
                                   [beanName]  =>  read
                            )
              )
)
```

12.3 An introduction to beans

Before we explain the process of selecting processes in PAINT, we need to explain the different types of beans that exist. Don't worry if this doesn't make sense immediately, there will be a step-by-step guide to selecting and using a process later on in this guide.

There are three types of beans in PAINT; entity, search and façade:-



Entity beans

Entity beans are the guts of the application. Each bean defines a particular logical entity in the application; its fields, validation, business logic etc. All actions you perform within PAINT (and PureResponse) ultimately use entity beans.

You can recognise an entity bean by the name. All start with "bus_entity" and then the package name (where applicable) and finally the logical entity name. For example, the email entity within the campaign package is called "bus_entity_campaign_email".

You don't access entity beans directly, but instead operate on them via façade beans (see below). You will still need to understand entity beans though, because the field names of the data you send to PAINT must match the appropriate entity bean, and the data returned from any processes is grouped by entity bean. See the example responses above.

Facade beans

Façade beans are the face or gateways of PAINT. You are only able to call processes that exist on façade beans and they are the only way that you can manipulate the data in the entity beans.

Note: When you use PAINT you'll get used to seeing the word "façade" a lot. The term has been taken from a design pattern that groups several server-side actions together for efficiency and simplicity.

The naming convention for façade beans is the same as that for entity beans but prefixed with "bus_facade" instead. What's more, facades are generally partnered to an entity bean and contain a default set of available processes e.g. create, load, update, store, remove etc. Because of this, it

should be relatively straight forward to work out what façade to use and what process to call, once you've chosen the entity bean. If the entity you want doesn't have a matching façade, then that means that either it is only manipulated via another entity (i.e. parent entity) or it is not available for manipulation via PAINT.

When sending a request to PAINT, you are asked to send an entity data parameter. This is the data that maps to the entity being updated. However, there is also an option to send "process" data. This data is used exclusively by the façade, and will be explained in the description of the façade process in the data dictionary.

Search beans

The final bean type is a search bean. These beans are, like façades, partnered to an entity bean. They define the parameters with which you may search for the partnered entity and contain a standard set of data to give you the results of the search.

The naming convention for a search is the same as for a façade or entity, but this time prefixed with "bus_search".

To access a search you must call the "search" process on the façade bean. To pass search parameters you must populate the entity data parameter with data that matches one or more of the fields on the search bean. This data will be validated and then the search will be performed.

The data returned from a search includes a list of "id" data for all results found. This data will be sufficient for you to load the individual entities. You can determine what data is required to load an entity by looking at the fields on the entity's matching "key" bean. These are located in the same package as the entity bean within the data dictionary.

To load the individual bean you can pass the individual id data element as the entity data parameter to the relevant "load" process on the façade.

12.4 Putting it all together in a session

The first thing you'll need to do when you use PAINT is to log in. After that you'll probably want to string several different requests all together. You can consider this to be a session and a session can exist across lots of different requests and even have gaps in between e.g. a series of user interactions.

Any session must start with a request to log in. Without this you will not establish a context for the other requests and they will return either security or system exceptions.

Logging In

The login process is found in the context façade bean ("bus_facade_context"). This is the one process that does not need to receive a context id in the request. Instead it must receive the user name and password of the user logging in.

> Note that a dedicated 'system' login should be used for all API calls. We will set-up and configure this upon request for you before you start working. If you login with a user name that is being used through any other interfaces then any other user currently logged in with the same user name will be logged out.

The following is a code snippet that shows user "abc" logging in with the password "123". After you've successfully issued this login request, the returned data will contain the context id for this user and this session. You'll need to store this context id and pass it back to us in all your other requests within this session.

```
// Establish the context for this send (login)
$connectArgs = array(

       "userName" => "abc",
       "password" => "123");

$resultData = soapHandleRequest(

       null,
       "bus_facade_context",
       "login",
       $connectArgs,
       null);

// Get the unique context reference for use in later calls
if($resultData["result"]=="success")
{
```

Don't forget to check the "result" value returned. If it isn't "success" then the login request has failed and you'll need to decide how to handle that response.

Log out

After the set of requests has been completed (or failed), you should issue the request to log out as the following code snippet demonstrates. Notice that you have to pass the context id back to us so we know who we're logging out.

Try to make sure you log out even if errors occur. This prevents you leaving data lying around on the system, and orphaned sessions.

```
soapHandleRequest(

       $contextId,
       "bus_facade_context",
       "logout",
       array(),    // No parameters
       null);
```

12.5 Choosing and calling a process (step-by-step)

Please refer to the full data dictionary files in the pack which can be found in:

`...API Developer Pack\Data Dictionary\index.html`

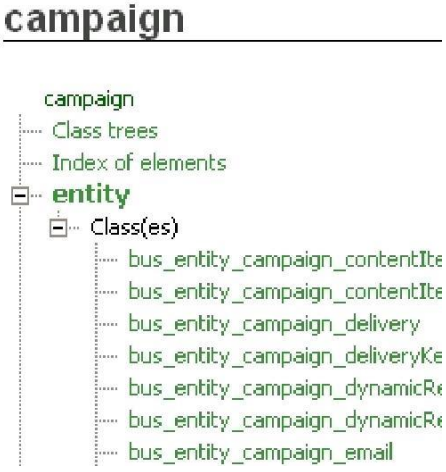
Step 1: Choosing your identity bean

The first step is to decide which entity you want to update. Open the data dictionary and you'll see that there is a drop down list of packages in the right-hand corner. Packages are used to group beans together based upon what the beans do.

Here is a summary of the packages available.

Package Name	Description	Examples
Campaign	Beans related to the campaign management process.	Email, list.
Account	Beans related to the logged in account settings from the group down to the login.	Pool (group), identity (profile).
Context	Beans that contain information about the current logged in session.	Context, preferences.
Paint	Framework beans that hold the common fields and methods for each type of bean.	Façade, entity, search.

In this example we want to use email. So select the campaign package and expand the entity section in the tree view on the left hand side.



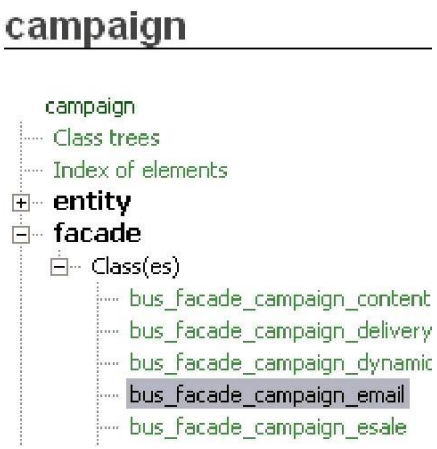
Here you will see each of the different entity beans within the campaign package. You'll also see that each bean has a "Key" bean associated with it, but we'll come to that later.

We know that we want to create an email, and can see that there is an email bean so select it and read the description. Since the description matches what we want to achieve, we have confirmed that this is the entity we want to update.

Step 2: Choosing your process

Now we have selected the entity we can move on to the façade bean and check which processes are available for it.

Select the façade section in the tree view on the left hand side and select the email façade.



You'll notice that the façade has some methods available on it.

You'll also notice that these methods are quite specific to certain actions related to emails and don't have the basic processes such as create, load etc. The reason for this is that these standard processes are held on the parent bean. Scroll down to the bottom of the page and you'll see a section holding the inherited methods.

Inherited Methods

Inherited From bean_facade

bean_facade::__construct()
bean_facade::clean()
bean_facade::create()
bean_facade::load()
bean_facade::reload()
bean_facade::remove()
bean_facade::search()
bean_facade::setInputData()
bean_facade::store()
bean_facade::update()

You'll see all of the standard methods that you might want to use. Here are some of the main ones described:-

Method Name	Description
Create	Create a new and empty entity bean and pass back a bean id with which to reference it. The empty bean is then held in the context and can be referenced, updated, and saved.
Load	Load an existing entity bean. This will require one or more key fields to be passed in. You can find the key fields by looking at the key bean for the entity e.g. for email it's bus_entity_campaign_emailKey.
Store	Once you have loaded an existing bean or created a new bean you can pass revised data into this process and it will be validated, saved, and then cleared from the session.
Remove	If you have loaded an existing bean then you can call this method to remove it from the system. Don't forget that you need a bean id for this and you'll only get this from using the load process first.
Search	Call this method to perform a search using the data mapped to the fields found on the entity's search bean. You can access the search bean details by expanding the search section in the left-hand tree view. For each bean that is returned from the search, you'll receive a set of key data that you can then use in the load

	process.

Since we want to create a new email, we will have to use the create process first, and then populate and save using the store method.

Step 3: Choosing your fields

Now we've decided what entity to create and what processes to call, the final step in the decision making process is to select the fields that we want to update on the entity.

The first process we're calling is the create process. This doesn't need to receive any data and will simply return us a bean id for the empty bean we create. We'll have to pass this bean id back in when we call the store process.

To decide what fields to populate in the store process, select the email entity bean again and scroll down to the variable summary.

Here you can browse through all the fields and see what they do. Click on the field to read the description, data type and other information. Note that some fields are read-only and so you can't update these. They will be marked with "readonly: true" as per the following example:-

Once you have decided upon the fields, you'll have all the information you need to begin putting it together:-

- The bean you're updating
- The processes you're calling
- The data you're passing

You'll then need to repeat this process for each different operation you want to implement via PAINT.

Special characters

You may find that you want to send us special characters in your XML. This might be foreign characters in custom data for list upload for example. If there is a possibility that you'll be sending us anything unusual in one of your fields then please use our special base 64 field names. This isn't too exciting, just append the string `_base64` to the end of the name of any field you send us and we'll interpret that as base 64 when we receive it, and decode it. It should save you some unwanted XML issues.

12.6 What next?

You should have enough information now to log in and log out, select your required processes and implement them on your platform. If you have problems during this process and are unable to achieve what you want to achieve, then the next step is to come back to us and discuss your requirement so we can provide you with the assistance you require.